Department of Computer Science

Technical Report

Undergraduate Dissertation: Adapting a Video Game to do Psychological Experiments

Jack Kenyon

Technical Report 2013-09 **November 2013**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Adapting a Video Game to do Psychological Experiments

Jack Kenyon

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2011

Adapting a Video Game to do Psychological Experiments

Adapting a Video Game to do Psychological Experiments

Submitted by: Jack Kenyon

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Abstract

Transitive Inference is a process of reasoning that has in the past been demonstrated to occur in various animals - including monkeys, rats and pigeons - as well as humans, including young children. This project documents the development of a piece of software for conducting experiments into Transitive Inference in humans. The software was in the form of a video game modification and developed using the Unreal Developers Kit, based on the Unreal engine. Several experiments were conducted to demonstrate that the software could, in future, be used to gather valid scientific data.

Contents

CONTENTS ..... **I**

LIST OF FIGURES ..... **IV**

LIST OF TABLES ..... **V**

ACKNOWLEDGEMENTS ..... **VI**

INTRODUCTION **1**

1.1 TRANSITIVE INFERENCE 1
1.2 PROJECT GOAL 2
1.3 A NOTE ON TERMINOLOGY 2

LITERATURE SURVEY **3**

2.1 BACKGROUND 3
2.2 PREVIOUS EXPERIMENTS 4
2.2.1 McGonigle and Chalmers (1984) *5*
2.2.2 McGonigle and Chalmers (1992) *5*
2.2.3 Davis (1992) *5*
2.2.4 Siemann and Delius (1993) *6*
2.2.5 Greene et al (2001) *6*
2.3 COMMENTS ON PREVIOUS EXPERIMENTS 6
2.4 IMPLEMENTATION METHODS 8

REQUIREMENTS **9**

3.1 REQUIREMENTS GATHERING 9
3.2 REQUIREMENTS SPECIFICATION 10
3.2.1 Functional System Requirements *10*
3.2.2 Non-Functional System Requirements *11*
3.2.3 Non-Functional Project Requirements *11*
3.3 ANALYSIS OF REQUIREMENTS 11

DESIGN **13**

4.1 INTRODUCTION 13
4.2 INITIAL DESIGN DECISIONS AND PROBLEMS 13
4.3 CRITIQUE OF THE DESIGN STAGE 15

IMPLEMENTATION **16**

5.1 INTRODUCTION 16
5.2 BUILDING THE LEVEL 16
5.2.1 Main Test Chamber *16*
5.2.2 Moving the Doors *17*

5.2.3 Triadic Test Chamber ... *18*
5.3 KISMET ... 19
5.3.1 Introduction ... *19*
5.3.2 Global Variables ... *19*
5.3.3 Determining Whether the Player has Chosen the Correct Door ... *21*
5.3.4 Guiding the Player Through the Training Phase ... *22*
5.3.5 Choosing the Next Pair During the Pair Testing Phase ... *23*
5.3.6 Updating the Output Variables ... *23*
5.3.7 Changing the Colours of the Doors After Each Choice ... *24*
5.3.8 Outputting the Test Results to an External File ... *25*
5.3.9 Triadic Testing ... *25*
5.3.10 Allowing the Experiment to be Customised by the User ... *25*
5.4 UNREALSCRIPT AND CONFIG FILES ... 27
5.4.1 Introduction ... *27*
5.4.2 Custom Kismet Action Classes ... *27*
5.4.3 Other Changes ... *29*
5.5 CRITIQUE OF IMPLEMENTATION ... 30

TESTING AND RESULTS ... 31

6.1 INTRODUCTION ... 31
6.2 TESTING MOVEMENT TO THE TRIADIC PHASE ... 31
6.3 TESTING CUSTOMISATION OF THE TRAINING PHASE ... 32
6.4 PLAYER TESTING ... 33
6.5 EXPERIMENTAL RESULTS ... 34

CONCLUSIONS ... 37

7.1 PROJECT SUMMARY ... 37
7.2 LIMITATIONS AND POSSIBLE EXTENSIONS ... 38
7.3 CRITIQUE OF THE PROJECT ... 39
7.4 CLOSING REMARK ... 40

BIBLIOGRAPHY ... 41

USER DOCUMENTATION ... 42

A.1 INSTALLATION ... 42
A.2 STARTING AN EXPERIMENT ... 42
A.3 THE DEFAULT EXPERIMENT ... 43
A.4 MODIFYING THE EXPERIMENT ... 43
A.5 COLLECTING THE RESULTS ... 44

RAW RESULTS OUTPUT ... 45

B.1 SUBJECT ONE ... 45
B.2 SUBJECT TWO ... 45
B.3 SUBJECT THREE ... 46
B.4 SUBJECT FOUR ... 46

CODE ... 47

C.1 SEQACT_SAVERESULTS ... 47
C.2 SEQACT_PARAMETERS ... 50
C.3 INFERENCEINFO ... 52
C.4 MYPLAYERCONTROLLER ... 52

C.5 MyPawn 53
VIEWING KISMET AND MODIFYING THE SOFTWARE 54

List of Figures

FIGURE 1. A SCREENSHOT OF THE LEVEL CREATION INTERFACE, SHOWING THE TEST CHAMBER 17
FIGURE 2. KISMET SEQUENCE CONTROLLING DOOR ANIMATION 18
FIGURE 3. SCREENSHOT OF TRIADIC TEST CHAMBER 19
FIGURE 4. GLOBAL VARIABLES, AS STORED IN KISMET, WITH THEIR INITIAL VALUES 20
FIGURE 5. KISMET SEQUENCE DEALING WITH PLAYER'S CHOICE OF DOOR 21
FIGURE 6. KISMET SEQUENCE USED TO UPDATE 'NO' OUTPUT VARIABLES 24
FIGURE 7. KISMET SEQUENCE USED TO REPAINT THE DOORS 24
FIGURE 8. AVERAGE PERCENTAGE CORRECT RESULTS FOR EACH PAIR (TRAINING PAIRS IN PARENTHESIS) 35
FIGURE 9. AVERAGE PERCENTAGE CORRECT RESULTS FOR EACH TRIAD 36

List of Tables

TABLE 1 - TRAINING SCHEDULE USED IN MCGONIGLE AND CHALMERS' 1984 EXPERIMENT 7
TABLE 2 - RELATIONSHIP BETWEEN CURRENTPAIR VARIABLE AND PAIR/TRIAD 21
TABLE 3 - TESTING SCHEME USED TO TEST USER CUSTOMISATION 32
TABLE 4 - REVISED TRAINING SCHEDULE USED DURING USER TESTING 34

Acknowledgements

My thanks to Dr Joanna Bryson, my supervisor for this project.

Thanks also go to Izzi, for being generally amazing over the year.

Chapter 1

Introduction

1.1 Transitive Inference

Transitive Inference (TI) is a process whereby one reasons that if some item A is greater than some other item B, and if B is greater than another item C, then A is greater than C. In other words, if A>B and B>C, then A>C. In some cases, such as with integers, this relationship is obvious and holds for all items in the domain. In other cases this is not so, for example with colours, where no one colour is greater than another unless placed in some arbitrary order. (Bryson and Leong 2007)

Transitive Performance (TP) is a similar process, except that it makes no claims regarding the mental processes that are used to perform the inference. This is shown when test subjects successfully infer that for example A is greater that C, but cannot explain how they know this. Studies conducted into Transitive Inference using adult humans have shown that, while some test subjects construct explicit mental models of the inference problem and then use those models to successfully solve inference tasks, other test subjects can perform equally well in the tasks, but when asked could not identify what the problem had been about, or even how they solved it (Siemann and Delius 1993).

Studies investigating Transitive Inference and Performance have been conducted for many years. Piaget was influential in this area and discussed the ability of small children to perform inference tasks as far back as 1928. McGonigle and Chalmers more recently performed inference experiments using monkeys (1992) as well as children (1984), and since the 1970s many other researchers have conducted inference experiments using many different species.

1.2 Project Goal

The goal of the project was to create a piece of software that could be used to conduct Transitive Inference experiments on humans. It was decided that this should be through a video game modification (henceforth shortened to mod) in order to help increase interest from test subjects and to provide a computer science basis for the project. After the software was completed, a possible goal was to conduct statistically significant scientific experiments on undergraduates. However, the software itself was the primary goal, as any potential future experiments could be conducted independently.

1.3 A Note on Terminology

Throughout this document a distinction has to be made between test subjects and those actually running the experiments. Throughout, the word 'user' refers to those running the experiments, while those being tested are referred to either as 'subjects' or 'players'.

Chapter 2

Literature Survey

Within this project there were several areas that had to be investigated before work on the project could commence:

- An understanding of exactly what Transitive Inference is and why it is worth investigating.
- Previous experiments carried out on Transitive Inference were investigated in order to ensure that the system follows similar experimental guidelines and produces useful data.
- Possible implementation methods had to be investigated and evaluated to determine which was the most suitable to the project.

2.1 Background

Transitive Inference and Performance as defined in the introduction are an important area of psychological investigation, especially the area of unconscious reasoning demonstrated by the existence of Transitive Performance. TP is important as it shows that humans can perform unconscious acts of deduction with equivalent success to those who consciously attempt to solve a problem of comparing items. (Siemann and Delius 1993)

Transitive Performance has also been observed to occur in young children (McGonigle and Chalmers 1984) as well as several animal species, including monkeys (McGonigle and Chalmers 1977) and rats (Davis 1992), i.e. those who are not normally considered to have deductive capabilities.

The majority of papers that were found on Transitive Inference put forward various theories and explanations for how subjects learn and complete TI and TP tasks. Such

discussions have been determined to lie outside the scope of the project and more in the realm of psychology than computer science, so such theories shall not be discussed here. What must be discussed, however, are the various experimental techniques used in those studies, as the project goal is to replicate these experiments and hopefully even allow further data to be gathered.

2.2 Previous Experiments

In this section of the literature review, several experiments previously conducted into Transitive Inference will be summarized.

Experiments into TI tend to follow an established pattern. Five (or more) objects are used, and ranked in some arbitrary order. This order sometimes changes depending on the test subject, either randomly or by having the order reversed. The ordering is generally denoted as the hierarchy (A>B>C>D>E). What these objects are, and how they are ordered, differs from experiment to experiment, largely based on who the experiment is designed for. A and E are referred to as edge objects.

In general, the experiments themselves are divided into two phases, the training phase and the testing phase. In the first phase subjects are presented with 'Training Pairs', that is pairs of objects that are adjacent to each other in the hierarchy. These Training Pairs are generally denoted as (AB, BC, CD, DE). The positions of the two objects are unrelated to the correct choice, i.e. the correct object is not always on the left/right side. Subjects are instructed to choose one of the objects, and are rewarded if they choose the object that is greater in the hierarchy. Depending on the study, the subject may be punished if they select the lesser object, or they may instead just receive no reward. This training continues across all Training Pairs until the subject correctly chooses the greater object to some criteria.

The testing phase usually follows immediately, to minimize the problem of a subject simply forgetting the training. This phase generally involves adding new 'Transitive Pairs' that are not adjacent into the sequence, but how this is done differs depending on the experiment. The BD pair is usually considered the most important, for various psychological reasons. The reward schedule for the testing phase varies greatly between experiments, either continuing to give rewards only for correct choices, rewarding the subject regardless of choice, rewarding the subject randomly or simply providing no reinforcement whatsoever.

Overviews of several previous inference experiments follow to highlight differences in technique.

2.2.1 McGonigle and Chalmers (1984)

The subjects were a number of young children, around 6 years old. The objects used for testing were tins painted one of five colours. The subjects were presented with pairs of these tins and rewarded with a counter hidden under the correct tin, which could be used after the experiment to 'buy' sweets. The training was also verbally reinforced by the examiner, as subjects were told when they were right or wrong.

The test phase consisted of blocks of all ten possible pairs. These pairs included the four Training Pairs, which were still only rewarded when a correct answer was given, as well as the six Transitive Pairs that are not adjacent. These six Transitive Pairs were rewarded regardless of what the subject chose.

This experiment also incorporated triadic testing, where the subject is presented with three choices instead of two. The triadic testing was conducted in a similar manner to the previous testing phase, simply replacing the Transitive Pairs with triads.

2.2.2 McGonigle and Chalmers (1992)

The experimental design of this study was virtually identical to that of the previous study, except that the subjects were monkeys. Subjects were presented with tins of different colours and rewarded with peanuts for successful choices. Triadic testing was again present in this experiment.

2.2.3 Davis (1992)

The subjects in this experiment were four female rats. In this case the stimulus involved smell instead of sight. Pairs were presented in the form of two tunnels, each of which had a different scent. Rats were rewarded with food if they selected the correct tunnel.

The testing phase focused mainly on the BD pair, as previously stated. In this case, neither choice was rewarded during testing.

2.2.4 Siemann and Delius (1993)

The subjects in this experiment were twenty four adults with a mean age of 23. The experiment was presented in the form of a video game, making this especially relevant to my project. In this case, there were six objects in the hierarchy, A>F, presented as six different polygons.

The 'game' involved leading a cartoon figure around a castle, successively choosing between two doors. Each door displayed one of the polygons, and was chosen by pressing either a left or right key. Subjects were rewarded for correct choices by receiving a coin and were punished by having to give a coin away to a beggar, before moving on to the next choice. Unusually, towards the end of the training phase random pairs were not reinforced at all (by leading the subject to an empty corridor) to prepare the subject for the unreinforced Transitive Pairs. These Transitive Pairs consisted of the pairs BD, BE and CE, i.e. new pairs that did not contain one of the edge objects. These pairs were unreinforced using the empty corridor, and mixed randomly with Training Pairs during the testing phase. The subjects took 35 to 80 minutes to complete the entire session. No triadic tests were involved.

2.2.5 Greene *et al* (2001)

The subjects were forty one students, roughly half of which were informed as to the nature of the experiment, half of them not. In this case, the objects were five characters from the Japanese Hiragana script. Subjects were presented with two characters on a screen, and had to select one of the two using one of two keys on a keyboard. Feedback came in the form of either the word “correct” or “incorrect” appearing on the screen. In this case the subjects received no reinforcement during the testing phase, including for the Training Pairs that they received. No triadic tests were involved.

2.3 Comments on Previous Experiments

There are obviously many variances in experimental procedure that had to be considered when designing the software. The number of objects used in the ordering may change, with a sequence of five objects being the most common, while Siemann and Delius used six. The reward schedule used during the testing phase differs often. In the Greene *et al* paper even the Training Pairs stopped receiving feedback during the testing phase. Siemann and Delius took the unusual step of preparing subjects for a lack of feedback during the latter part of the training phase. McGonigle and Chalmers included triadic testing, while the rest did not.

In terms of how the subject is trained there are also variances, mostly regarding the order that the pairs are presented to the subject. McGonigle and Chalmers (1984) first presented each pair in order until the subject demonstrated that they understood the correct choices. i.e they were given the first pair DE repeatedly until 9 out of the 10 most recent trials were correct, before moving onto CD, and so on. After all pairs were trained successfully, there was a stage of 'runs' of trials, which present each pair several times before moving onto the next pair. If the subject responds well, each pair is shown less times in each run, until the runs are reduced to one trial per pair. Randomly ordered runs of pairs completed the training. After most of the training stages, subjects were rejected from the experiment if a certain level of understanding was not reached. The procedure used in the 1984 experiment is summarized in the following table, taken from Bryson and Leong (2007)

Table 1 - Training Schedule used in McGonigle and Chalmers' 1984 Experiment

Stage	Training and criteria
S1	Each pair in order (ED, DC, CB, BA) repeated until 9 of the 10 most recent trials correct. Reject if requires over 200 trials total.
S2a	4 of each pair in order. Criterion: 32 consecutive trials correct. Reject if requires over 200 trials total.
S2b	2 of each pair in order. Criterion: 16 consecutive trials correct. Reject if requires over 200 trials total.
S2c	1 of each pair in order. Criterion: 30 consecutive trials correct. No rejection criteria.
S3	1 of each pair randomly ordered. Criterion: 25 consecutive trials correct. Reject if requires over 200 trials total.

In contrast, Siemann and Delius (1993) randomly ordered their trials from the beginning. S1 was also replaced by randomly ordered groups of eight, which later reduced to groups of four and so on. Criteria for advancing to the next stage of training were also different. In this case the training moved onto the next stage every 80 trials, reaching the stage of randomly sequenced individual trials from trial 241 onwards. After this point the subject was deemed to have completed training upon reaching an accuracy of >70% on every training pair within a block of 60 consecutive trials.

Greene *et al* (2001) used four stages of forty trials in a standard order until the last stage, where randomly sequenced trials were again used. Success criteria in this case was 80% accuracy on each pair, and rejection instead meant repeating that training stage. Davis (1992) used a differently ordered sequence of training pairs in order to minimize confusion for his rats.

As can be seen from even these four experiments (McGonigle and Chalmers used a similar training system for both monkeys and children) there are many possible ways

to train a test subject to perform a Transitive Inference task. These variances have to be taken into account when building the system. Individual experiments may require different strategies for training, so the ability of a user to alter the training regime should be incorporated into the software to make it as useful and re-usable as possible.

2.4 Implementation Methods

Two possible options had previously been identified for development of the system. One was the UDK, or Unreal Development Kit. This is a free development environment used to create video games based on the Unreal Engine 3, which has been used to create a wide variety of video games in the past. The Unreal Engine 3 has its core written in C++, however most of the gameplay code can be written in UnrealScript, an object-oriented scripting language built specifically for the Unreal Engine.

During the investigation of the UDK, several features were found that made it an attractive option for the task at hand. The 'Level Streaming' functionality allowed by the UDK seemed like it would be helpful in moving the player between trials, although this eventually proved unnecessary. The built in 'Kismet' scripting tool looked to be extremely useful for the task, allowing the logic of the system to be built and represented in a clear way. Using UnrealScript also promised to allow the recording and output player data, i.e. test results.

The other identified option was the Source SDK, based on the Source Engine, which has been used in video games such as Half Life and Portal. The SDK is mostly focused on producing video game mods, and also uses C++. As such it could almost certainly be used to create an appropriate system for the project. However, the built in tools for level creation (the Hammer Editor) and scripting were deemed by the author to be more complex and less user friendly, in return for additional functionality not necessary for this project. Therefore, the UDK was chosen as the development environment.

Chapter 3

Requirements

3.1 Requirements Gathering

Requirements were gathered by analysing previous inference experiments and by discussions with Dr Bryson. Specific requirements were then determined based on these. The functional requirements provided a generic outline of the chosen experimental procedure while leaving specific details such as the training criteria unspecified. This is to allow those conducting any experiments to customise the precise way their experiment is run.

Throughout, certain words are used to indicate the importance of the requirement in question. These are defined as:

- '**Must**' indicates that the requirement is an absolute necessity. The system must satisfy the requirement to be considered a success.
- '**Should**' indicates that the requirement is considered very useful, but not mandatory. While it would be preferable to satisfy the requirement, the system will still be usable without it.
- '**May**' indicates that the requirement would be useful but is not particularly important. In other words, optional.

3.2 Requirements Specification

3.2.1 Functional System Requirements

1. The software must be in the form of a video game.
   1.1. This game must be built using the Unreal Developers Kit (UDK).
2. In this game, the test subject must be presented with a series of pairs of doors, and will have to choose which door to move through each time.
   2.1. The position of the correct door is randomly chosen, and is equally likely to be the left or right door.
3. There must be five possible colours of door, and these colours will have an internal ranking unknown to the subject.
4. The game must start with a training phase, where the subject will be taught the five Training Pairs.
   4.1. In the training phase, if the subject selects the door with a higher ranking, they must be 'rewarded' by the game in some way.
   4.2. In the training phase, if the subject selects the door with a lower ranking, they must be 'punished' by the game in some way.
5. After the subject has demonstrated that they can select the correct door in for each Training Pair to some criterion they must be moved to the pair testing phase.
   5.1. In this phase, the subject must be presented with several blocks of all ten possible pairs in a random sequence.
   5.2. For the Training Pairs, the subject must be rewarded and punished as above.
   5.3. For the new Transitive Pairs, the subject must be rewarded regardless of the door chosen.
6. After several testing blocks, the subject must be moved on to the triadic testing phase, and presented with three doors to choose between.
   6.1. In this phase, the subject must be presented with several blocks of all ten possible triads in a random sequence.
   6.2. As in the pair testing phase, the subject must be rewarded regardless of the door chosen.
7. As the subject is playing the game, the software must collect data detailing how successful the subject has been. This data must be output in a form readable to humans when the experiment is concluded.

7.1. This data must include the success and failure rates of the subject on the different pairs/triads used.

7.2. The data may include the time the subject takes to complete each trial.

7.3. The data may include the time the subject takes to complete each training phase.

8. Those conducting the experiment should be able to modify certain parameters to enable customisation of the experimental procedure.

8.1. This should include parameters relating to the schedule used during the training phase, and should allow the user to skip stages of training entirely.

8.2. This should also include the number of test blocks given to the subject during both the pair testing phase and the triadic testing phase, and should also allow the user to skip triadic testing entirely.

8.3. This may include the number of colours used in the ordering.

3.2.2 Non-Functional System Requirements

9. The system must be thoroughly tested and have a minimum of bugs, as any bugs encountered by the subject would invalidate the test results for that subject.

10. The game must run at a steady frame rate, so as not to distract the subjects.

11. The game should be intuitive to control, so that the subject may focus on choosing doors and not on learning the control scheme.

12. The system must run on modern Windows machines.

3.2.3 Non-Functional Project Requirements

13. After developing the software system, the experiment must be conducted on several test subjects in order to demonstrate that the system works and can be used to collect experimental data.

14. The system should be completed by the 10th of April (Final hand in on the 10th of May.

3.3 Analysis of Requirements

The process of gathering requirements was not done according to any specific plan or method, and the functional requirements essentially specify the experimental procedure that is to be implemented. There are several lower priority requirements

listed, specifically 7.2, 7.3 and all parts of 8. 7.2 and 7.3 are related to gathering additional timing data from the experiment, and were not included in the final implementation. Requirement 8 relates to the ability of the user to customise the experimental procedure used, a function that was deemed important but not essential to the core functionality of the system. Subsections 8.1 and 8.2 were successfully implemented, while 8.3 was not, due to time constraints.

Requirements 5.3 and 6.2 merit some discussion. These specify that during the testing phases the subject is to be rewarded regardless of the choice made, regardless of whether they were correct. This as with most of the experimental design is based on McGonigle and Chalmers' experiments, with the aim of reducing learning during the testing phase. To paraphrase Bryson and Leong (2007) this is thought to be the least disruptive non-discriminative reward schedule, as the subject probably expects to be rewarded for their choice, and since learning is most likely to occur when expectations are violated it is less disruptive to meet those expectations and reward them anyway.

Non-functional requirements were rather sparse, as the success of the system largely hinged on whether it could fulfill the functional requirements, and any usability issues are largely inherent in the system. If a subject can use the arrow keys, they should be able to navigate the experiment. Requirement 12 specifies the use of Windows machines. This is due to limitations of the UDK, which only runs on Windows machines, and the difficulty of porting games such as this to other platforms.

Chapter 4

Design

4.1 Introduction

The initial design stage of the project was not very extensive, as at this point of the project the capabilities and functions of the UDK were not well understood. Formal design diagrams were not created; instead a plan of the system was used that was created from the requirements and refined during initial experimentation with the UDK and corresponding tutorials.

This unfamiliarity with the UDK lead to some issues later on, as some development time was lost pursuing implementation methods that were not useful or necessary. The design process was in many ways iterative, and the design of the system was modified and refined during implementation. This chapter will document those early design decisions and discuss some of the issues that resulted from them.

4.2 Initial Design Decisions and Problems

The first design decision to be made was the ordering to be used in the experiment. Five colours were chosen, based on their distinctiveness from each other, and placed in an arbitrary order. The final order chosen for use in the system was:

Red>Blue>Green>Yellow>Purple

Initially, the plan for the system involved the concept of level streaming. This allows multiple game levels to be connected together seamlessly by loading a level before it is encountered by the player. This was thought to be a useful idea for the purposes of the project, with the idea that a level would be created for each pair of colours, with the player able to move from one level to the next seamlessly.

The game would determine which pair is needed next based on the player's choice and quickly load the required level for the player to move into next. However, initial experiments into using this method revealed several problems:

- It was simply too slow. Despite the small size of the levels in question the time taken to load the next level was not acceptable and broke the flow of the experiment.
- This required many different levels to be created, all with only superficial differences. Any changes made to the Kismet for one level would require corresponding changes made to the other levels.
- There were several issues with tracking variables. As seen in Chapter 5, variables are contained within the Kismet tool, and moving to a new level would require transferring these values to the new level, a difficult task that would have required a large amount of unnecessary UnrealScript.

It was decided that the proposed method of level streaming was inadequate, and a new plan was created involving only one level and the use of a teleport function. This plan is documented in the next chapter.

Another concept that was considered and partly implemented before being abandoned is that of creating a custom gametype for use in the game. A gametype is an UnrealScript class that specifies the rules used in any particular game, and is written so that it can be used on any given game level. The use of a custom gametype is advised in many of the UDK tutorials available online, so using one was considered early on.

As it transpired during implementation a custom gametype was not needed, as Kismet proved perfectly adequate for fulfilling the requirements of the project. As the experiment would only ever use the one level it was perfectly acceptable to implement the rules for the game within that level's Kismet sequences.

Thankfully, there were more successful design decisions made at this stage. It was determined that a somewhat modular approach to the Kismet sequences would be appropriate. The sequences of events and actions that were required over the course of the experiment were divided into appropriate subsequences, as listed below in the order that they were implemented:

- Determining whether the player had chosen the correct door.
- Guiding the player through the training phase.
- Choosing the next pair during the pair testing phase.
- Updating the output variables (once the player has entered the testing phase).
- Changing the colours of the doors after each choice.

- Outputting the results of the experiment to an external file.
- Moving the player to the triadic testing phase, which again includes sequences for updating output variables, changing door colours etc.
- Allowing customization of certain elements of the experiment by the user.

These sequences made up a large part of the project, as documented in the next chapter. The use of modular design elements allows the system to be built up slowly and also allows each sequence to be tested thoroughly before implementing the next part. The modular design would also make maintenance and additions easier, as each subsequence can be changed without seriously impacting the rest of the system.

Over the course of implementation, this modular ethos was largely adhered to, although there were some points where a strictly modular implementation was not feasible and the sequences had to refer to each other in unexpected ways. This stage of design still proved very useful during implementation, providing clear goals throughout.

4.3 Critique of the Design Stage

The design of the system was not very formal, and initial plans were discarded in several cases during implementation when it was discovered that they were unnecessarily complex or simply not needed. This lack of a formal design stage was mostly due to inexperience with the UDK, and it was decided that even a high level plan of how to go about implementation would likely become obsolete as the capabilities of the UDK became more clear.

There were still benefits to the work done at this stage. Deciding on the modular approach to the Kismet sequences provided clear steps that had to be taken during that part of implementation, and resulted in a system that was relatively simple to modify or correct.

A more formal design stage would have likely saved some development time later on. In any future projects using the UDK or other similar development suites the knowledge gained during implementation would certainly be used to create a more formal set of design documentation. As it is, the lack of formal design did not have a tremendous impact on the rest of the project, as once these unnecessary design elements were identified and removed the implementation of the system went smoothly, and is explained in the following chapter.

Chapter 5

Implementation

5.1 Introduction

The system was implemented using the aforementioned Unreal Development Kit (UDK). This is a free piece of software released by Epic Games to allow for the development and release of independent video games using the Unreal Engine 3. There were three main sections of the implementation:

- Building the level itself.
- Using Kismet to create the internal logic of the level.
- Writing custom classes in UnrealScript and using Config files.

These three sections will make up the bulk of this chapter.

5.2 Building the Level

5.2.1 Main Test Chamber

The UDK provides robust tools for level creation. The first part of the level to be built was the main test chamber, seen in Figure 1. This chamber consists of three connected rooms, two doors, several lights and a player start. The doors are tied to triggers, invisible to the player, that make them slide open at the player's approach and close when the player has passed through. The player start is the point that the

player enters the level from. The doors have no colour by default, and instead have the relevant colours applied to them using Kismet.

This simple chamber was then used for the construction of the Kismet sequences, which will be covered in section 5.3.

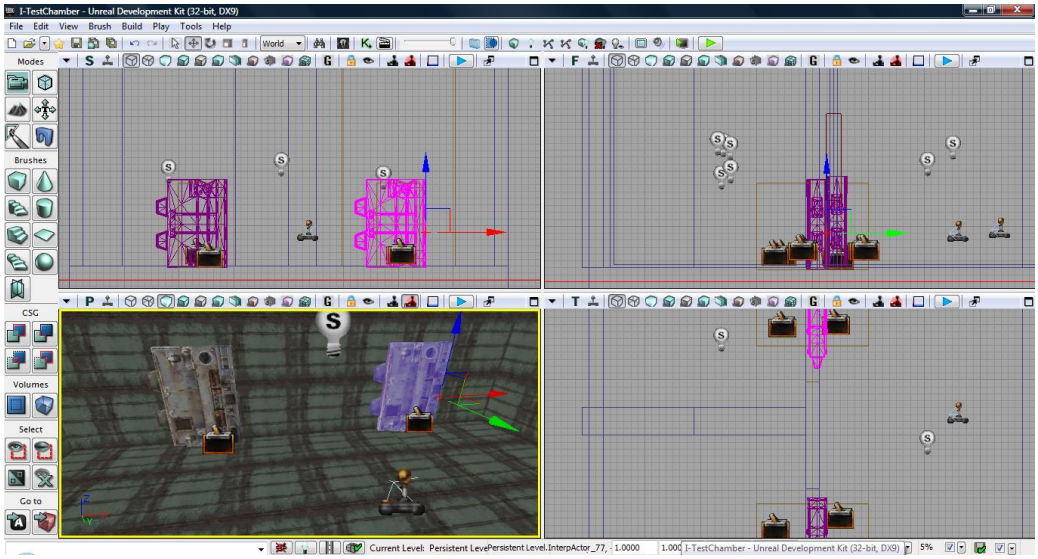

Figure 1. A Screenshot of the Level Creation Interface, Showing the Test Chamber

5.2.2 Moving the Doors

The doors' opening animation was created using Kismet and the 'Matinee' editor, and will be explained in detail in this section in order to give a brief introduction into how Kismet works.

Figure 2 shows the Kismet sequence that controls the animation of one of the doors. The sequence consists of five components:

- Two Events: Trigger_2 Touch and Trigger_3 Touch (The two red objects). These Events are activated when the player moves over the respective triggers.
- The Matinee Action and the Matinee Object that stores the specific animation data used (The two orange objects).
- The door Object that the Matinee Action operates on (The purple object).

These components interact in a hopefully obvious way. When Trigger_2 is touched by the player, the Matinee Action plays. This causes it to use the Matinee Object on the door Object, which in this case makes the door slide open. Then, when Trigger_3

is touched (on the other side of the door) the animation reverses, sealing the player inside of the room.

In the context of the larger system, Trigger_3's touch Event is also connected to several other things, notably the closing animation for the other door (to ensure that both doors are reset) and the set of commands to determine whether the player chose the correct door. Trigger_3 is also given a delay of 1.6 seconds after it is activated before it can be activated again. This is to ensure that it is not activated multiple times before the player is teleported away.

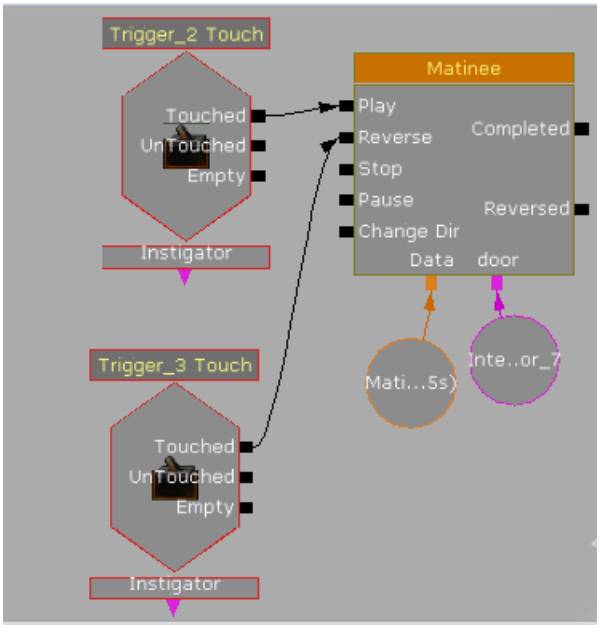

Figure 2. Kismet Sequence Controlling Door Animation

5.2.3 Triadic Test Chamber

After the Kismet sequence for the first chamber was completed and tested, a second test chamber was constructed. This chamber is placed adjacent to the first, and is for the purposes of triadic testing. Figure 3 shows a screenshot of the triadic test chamber. This chamber operates similarly to the main chamber, in that it consists of three doors that are opened and closed by trigger objects. All doors have identical Matinee Objects, as the animation required for them is the same in all cases.

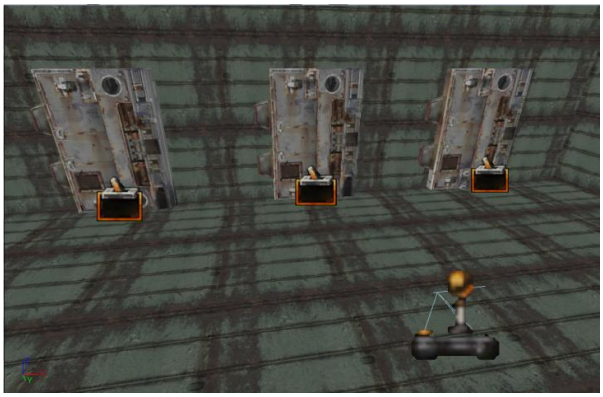

Figure 3. Screenshot of Triadic Test Chamber

5.3 Kismet

5.3.1 Introduction

Kismet is a powerful tool that allows the scripting of gameplay flow in a level. This is accomplished by connecting Sequence Objects together to form complex sequences. The use of Kismet became a large part of this project, and allowed the majority of the requirements to be implemented successfully. Several linked sequences were constructed in Kismet to perform many functions, corresponding to those identified and listed in section 4.2. The remainder of this section will go into further detail on these sequences. However, before any of these could be constructed variables had to be defined.

5.3.2 Global Variables

Several global variables were defined that are used throughout the Kismet sequences. While there are many variables (71) in the final system, the majority of these are used to either store the results of the experiment or to allow for user customization of experimental procedure, and will be discussed later. There are 8 variables that are used within the system itself, (shown in Figure 4):

Figure 4. Global Variables, as Stored in Kismet, With Their Initial Values

- Training? - used to store whether the player is still in the training phase
- CurrentPair - contains an integer representing which pair of doors are currently being used. This is later used to represent which triad is used during triadic testing. The relationship between the integer stored here and the pairs is shown in Table 2.
- WhichSide? - stores the location of the correct door. 0 means that the left door is correct, 1 means that the right door is correct.
- Stage - keeps track of which part of the training phase the player is in.
- SuccessTrials - stores the number of consecutive correct choices the player has made. Used in the training phase.
- TriadWinner? - used in the triadic testing phase instead of WhichSide?. In this case, 0 represents the left door, 1 the middle, and 2 the right.
- TrialsThisPair - used to decide when to move onto the next Training Pair. Also used during the testing phase to determine when each testing block is finished.
- TrialsThisStage - used to determine whether a player has failed training, when to move to the triadic testing phase, and when to end the experiment.

Table 2 - Relationship Between CurrentPair Variable and Pair/Triad

Value of CurrentPair	Pair (* indicates Training Pair)	Triad
1	Red>Blue*	Red>Blue>Green
2	Blue>Green*	Red>Blue>Yellow
3	Green>Yellow*	Red>Blue>Purple
4	Yellow>Purple*	Red>Green>Yellow
5	Red>Green	Red>Green>Purple
6	Red>Yellow	Red>Yellow>Purple
7	Red>Purple	Blue>Green>Yellow
8	Blue>Yellow	Blue>Green>Purple
9	Blue>Purple	Blue>Yellow>Purple
10	Green>Purple	Green>Yellow>Purple

5.3.3 Determining Whether the Player has Chosen the Correct Door

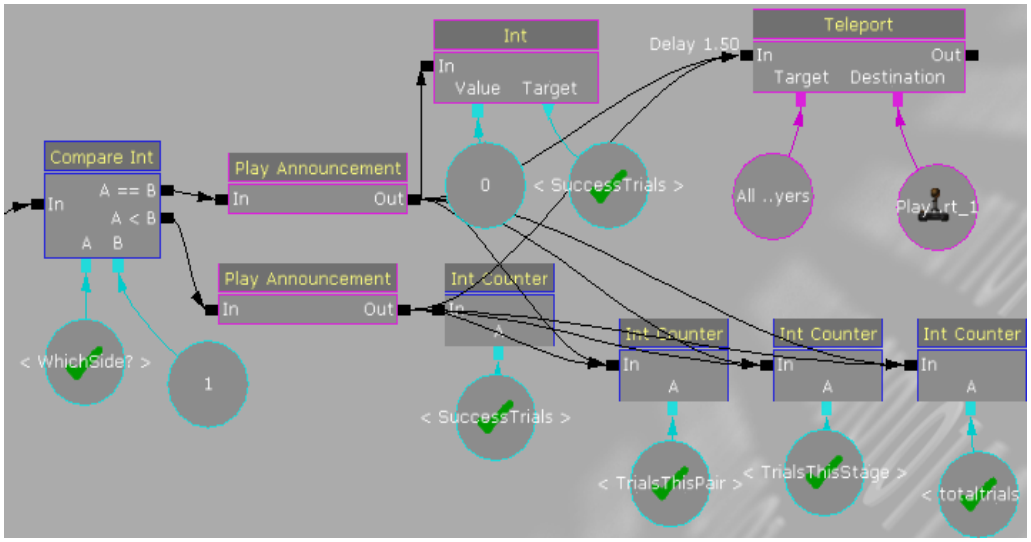

Figure 5. Kismet Sequence Dealing with Player's Choice of Door

The 'correct door' in this case is determined by the WhichSide? variable, which is set to a random value (either 0 or 1) at the beginning of every trial. As the sequence for actually showing the colours of the doors was not implemented at this point, the value of WhichSide? was set to display onscreen at the start of each trial to enable testing.

Figure 5 shows the Kismet sequence that is called when the player touches one of the two triggers (in this case the trigger behind the left door, although the two sequences are largely the same). When the trigger is touched, WhichSide? is checked to find out whether this was the correct choice. If it was (the lower sequence) an announcement is activated that displays 'Correct' on screen, with no audio feedback. SuccessTrials is then incremented.

If the incorrect choice is made then the higher sequence is taken, activating a different announcement (that displays 'Wrong' onscreen) and resetting SuccessTrials back to 0. In both cases TrialsThisPair, TrialsThisStage and TotalTrials are incremented, and after a delay of 1.5 seconds the player is teleported back to the start point, ready for the next choice.

In the original implementation choosing the correct door also offered audio feedback. This was discovered during testing to be distracting to the subjects, as it is encountered much more often, and was therefore removed.

5.3.4 Guiding the Player Through the Training Phase

The Kismet for this sequence is rather large and unwieldy, so screenshots shall be omitted in favour of a written description. This sequence is called by a 'Remote Event' Action placed before the Teleport Action from the previous section, and therefore activates after each trial. Initially this sequence triggers the random assignment for WhichSide?, and then performs a check to see if Training? is true. If so, it moves on to the main body of the sequence.

First, the value of Stage is checked to determine which stage of training the player has reached. The structure for each stage is largely the same, mostly varying in the values of the criteria required to progress (criteria which can be changed by the user in the final product). The general sequence for each stage is:

1. TrialsThisStage is checked to see whether the player has failed training (by exceeding the failure criterion, 200 by default).
2. If the player has not failed, SuccessTrials is checked to see whether the success criterion has been met.
3. If the player has met success criterion, the Stage variable is incremented, and certain other variables are reset to their initial value in preparation for the next stage.
4. If the player has not met success criterion, TrialsThisPair is checked to see whether the doors need to change colour. If so, CurrentPair is incremented (or looped back to 1 if required).

In stage 0 of training, this sequence is modified slightly in that the doors only change colour once the player has achieved criterion, and the next stage is entered after all Training Pairs have been completed.

In the final, random stage of training, this sequence is again modified by randomly changing CurrentPair after every trial. Also, instead of incrementing Stage after reaching success criterion, the Training? variable is set to false.

The success and failure criteria for each stage of the training can be customised by the user by editing Config files, as explained in section 5.3.10. The criteria for stage 0 notably differs from that used by McGonigle and Chalmers as specified in Table 1, in which the subject had to have 9 of the most recent 10 trials correct to move on to the next pair. As implementing this would require the system to keep track of the previous 10 trials during this stage the criterion was simplified. The subject instead has to have n correct consecutive trials to move onto the next pair, where n is specified by the user.

In this way the player is lead through the training phase. After every trial it is determined if the player has failed training, whether to progress to the next stage of training and whether it is necessary to change the colour of the doors.

5.3.5 Choosing the Next Pair During the Pair Testing Phase

During the testing phase, the extensive Kismet used during the training phase is no longer required, and the system simply uses a Random Switch object to randomly choose the next door encountered. This Switch resets after every 10 trials, meaning that all 10 pairs must be encountered before any pair repeats. This is the same construct used in the final stage of training, except with more pairs.

Additionally, during the testing phase the feedback given to the player is changed. The player is always given positive feedback when choosing between the new Transitive Pairs in accordance with requirement 5.3. The four Training Pairs continue to be reinforced accurately.

5.3.6 Updating the Output Variables

There are two simple sequences used to update the output variables during the pair testing phase. These are called when the player has activated a trigger behind a door, as used in section 5.3.3. Figure 6 shows the sequence that is called if the player's choice is incorrect. This uses a Switch object to read the value of CurrentPair and increment the corresponding 'No' output variable. A similar sequence is called if the player selects the correct door.

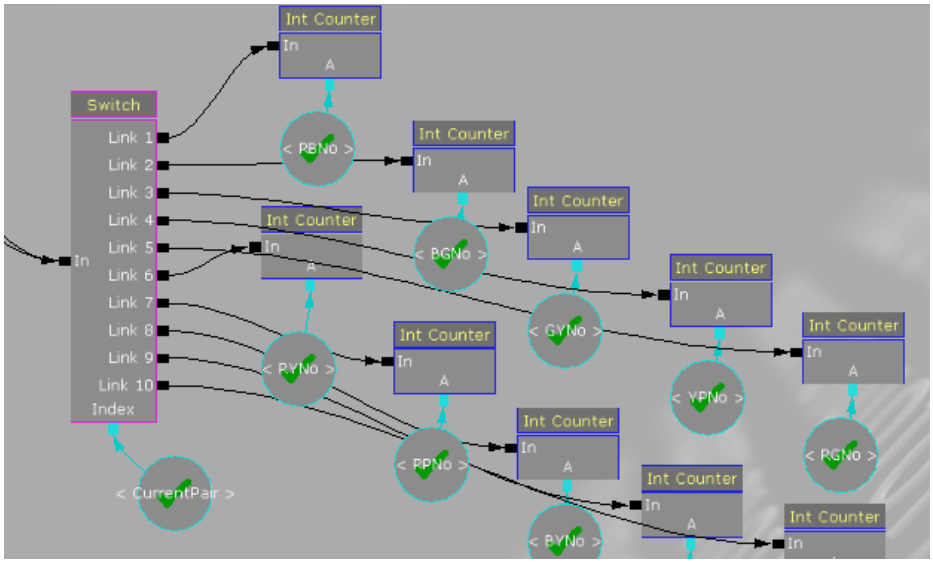

Figure 6. Kismet Sequence Used to Update 'No' Output Variables

5.3.7 Changing the Colours of the Doors After Each Choice

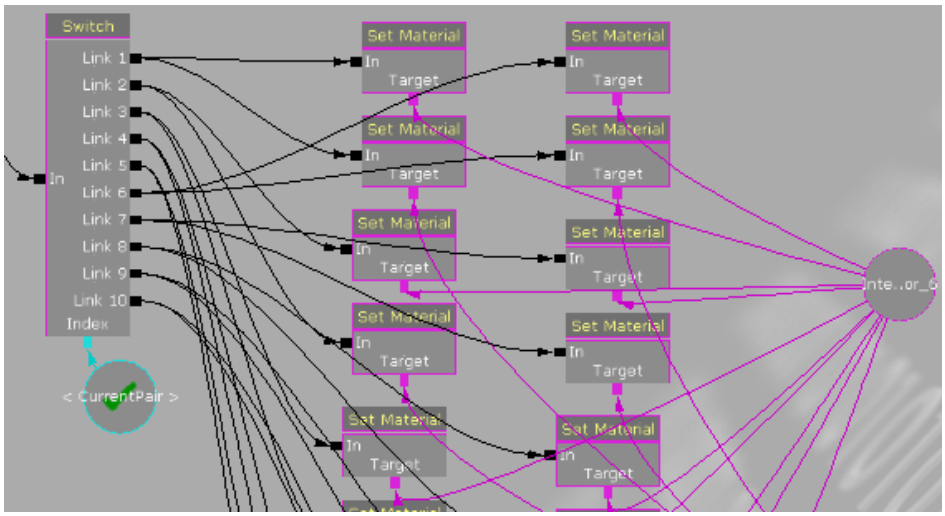

Figure 7. Kismet Sequence Used to Repaint the Doors

The doors are 'repainted' after every trial with the colours corresponding to the CurrentPair variable. Figure 7 shows part of one of two Kismet sequences used for this. At the start of this sequence WhichSide? is checked to determine the winning door. This check then leads to one of two Switch objects, as used in the previous section.

The Switch object reads CurrentPair and then 'repaints' the doors appropriately via the Set Material Action. This sequence is called after the Teleport Action in section 5.3.3, after all calculations regarding the next required pair have been made.

5.3.8 Outputting the Test Results to an External File

Kismet does not natively support this functionality, so a custom action had to written in UnrealScript, described in more detail in section 5.4.2. The new action, named Save Results, takes as input all of the output variables that are updated as the player moves through the game. When Save Results is called it updates an external Config file named UDKResults, which can be accessed after the experiment by the user. Save Results is called by a Remote Event, 'EndGame', once the player has completed the triadic testing phase or in the event that the player fails the training phase.

5.3.9 Triadic Testing

After the Kismet for the main test chamber was completed and tested, a separate section of Kismet sequences was constructed for use during the triadic testing phase.

During the pair testing phase, a check is performed before every teleportation to see whether it is time for the player to move onto the triadic testing phase. If so, a different Teleport Action is called that causes the player to move to the triadic test chamber. Once the player is moved to this chamber the original set of Kismet sequences is no longer used.

The triadic tests use a separate set of Kismet sequences that are essentially different versions of the sequences used in the previous chamber, remade to take into account the additional door. These consist of sequences handling the animation of the new doors, the random selection of the next pair, updating output variables and repainting the doors after each trial.

After each triadic trial TrialsThisStage is checked to see whether the player has finished the experiment. If so the EndGame Remote Event is triggered, causing the experimental results to be saved and the experiment to end. Ending the game itself is handled by using the console command 'disconnect' to send the player back to the main menu after they are told that they have completed the experiment.

5.3.10 Allowing the Experiment to be Customised by the User

This is achieved in a similar manner to that used in section 5.3.7. A custom class was written in UnrealScript that, when called via Kismet, reads in the values stored in a Config file named UDKExperiment and stores these values in Kismet variables. In

this way, the user is able to change these Kismet variables by editing the Config file. The variables are then used during the training phase by comparing them with the player's progress to determine when to move the player to the next stage. The customizable variables were chosen to be:

- Success criteria for each stage of training.
- Failure criteria for each stage of training.
- How many of each pair are needed before a colour change for each stage of training.
- Number of test blocks in the pair testing phase.
- Number of test blocks in the triadic testing phase.

This allows the user to customise the length of the training and the amount of experimental data gathered. All of these variables of course have default values based on those used in McGonigle and Chalmers' 1984 experiment, but it is acknowledged that others wishing to use the software may wish to have finer control of the experiment.

For the first two customisation options (success and failure criteria) it had to be recognised that the user may wish to not include failure criteria for a particular stage or just skip a stage entirely. This is accommodated in this case by using 0 as the value for one of these variables. The Kismet sequence was edited to check for 0 values as the player moves through each stage and will either not check for failure conditions or simply increment the Stage variable as required. Similar functionality is also provided for the triadic testing phase, allowing the user to end the experiment directly after the pair testing phase.

Two things are notably absent from this list:

- The ability to change the colours/patterns used on the doors.
- The ability to increase the number of colours used.

Both were originally considered for inclusion, but proved difficult to implement. The first is difficult because the colours are applied to the doors using the Set Material action, each of which has the colour to be used hard coded as a property. As these materials are created and stored within the UDK, any customisation of these would require the user to create the new material themselves and then edit the Kismet manually to refer to these new materials.

The second was not implemented mostly due to time constraints. The Kismet created for the system was built using 5 different colours. Adding the option for additional colours would require construction of entirely new sequences of Kismet for every additional colour. As a five object ranking seemed most common during the Literature Review, this was chosen for the system.

5.4 UnrealScript and Config Files

5.4.1 Introduction

UnrealScript is an object-oriented language based on C++ that allows more low level control of a game. Extensive use of UnrealScript was not required for this project, but it was used in two cases:

- Writing custom Kismet action classes to output experimental data and to allow user customisation.
- Modifying the game in other ways to streamline the experience.

In addition to UnrealScript, certain Config files had to be modified during implementation. Config files are .ini files that are used by the game to store certain settings, and can be accessed and modified by the use through any text editor. Config files are covered in more detail in section 5.4.3.

5.4.2 Custom Kismet Action Classes

Two custom classes were created for use with Kismet, SeqAct_SaveResults and SeqAct_Parameters. Truncated code listings follow:

Listing 1 - SeqAct_SaveResults

```
class SeqAct_SaveResults extends SequenceAction
Config(Results); //declares that the class uses the Results Config
                 //file

var() config int RBYes; //variable declarations
var() config int BGYes;
...

function Activated()
{
SaveConfig(); //when activated via Kismet, saves the input variables
              // to the specified Config file.
}

defaultproperties {
ObjName="Save Results" //declares the name of the action in Kismet
ObjCategory="Output"
// the rest creates the input/output/variable links used in Kismet.
InputLinks(0)=(LinkDesc="In")
OutputLinks(0)=(LinkDesc="Out")
VariableLinks(1)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBYes",
bWriteable=true, PropertyName=RBYes)
```

```
VariableLinks(2)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGYes",
bWriteable=true, PropertyName=BGYes)
...
}
```

This class, when used as an action in Kismet, reads in the values of the variables connected to it and then saves them in the Config file "UDKResults" (the UDK is added automatically).

Listing 2 - SeqAct_Parameters

```
class SeqAct_Parameters extends SequenceAction
Config(Experiment); //declares that the class uses the Experiment
                    //Config file

var config int PassZero; //variable declarations.
var config int PassOne;
...
var() int PZ;
var() int PO;
...

function Activated()
{
   PZ = PassZero;
   PO = PassOne;
   ...
}

defaultproperties {
ObjName="Parameters" //declares the name of the action in Kismet
ObjCategory="Output"
// the rest creates the input/output/variable links used in Kismet.
InputLinks(0)=(LinkDesc="In")
OutputLinks(0)=(LinkDesc="Out")
VariableLinks(1)=(ExpectedType=class'SeqVar_Int',
LinkDesc="PassZero", bWriteable=true, PropertyName=PZ) //creates the
input links used in Kismet.
VariableLinks(2)=(ExpectedType=class'SeqVar_Int',
LinkDesc="PassOne", bWriteable=true, PropertyName=PO)
...
}
```

In this class, two separate sets of variables were needed. The config variables contain the values stored in the UDKExperiment file, while the other variables are used in the Kismet action. Then, when the action is activated, these variables take on the values stored in the config variables (and therefore the Config file) and pass these on in turn to the Kismet variables, ready to be used.

5.4.3 Other Changes

As mentioned in Chapter 4, initially a new custom gametype named 'Inference' was created for use with this project. A gametype is a class that specifies the rules for a particular game. The default gametype, for example, is named 'Deathmatch'. The new Inference gametype along with the related classes 'MyPlayerController' and 'MyPawn' were created early on in implementation, in anticipation that it would be required for much of the system.

This proved to not be the case, as the capabilities of Kismet and the ability to modify existing files proved sufficient for the task at hand. Therefore in the final system these three classes are not actually used. They are still included in Appendix C and in the code submitted because they are still referenced in the rest of the system, despite not actually being used.

Several existing files were modified during implementation for various reasons. These included several Config files:

- DefaultEngine, changed to include references to the abovementioned Inference gametype.
- DefaultGame, as above, as well as some miscellaneous tweaks.
- DefaultInput, which stores the control scheme. Changed to unbind certain keys providing unhelpful functions (such as Feign Death) and to make movement within the game more intuitive by forcing the arrow keys to make the player strafe, instead of turn.
- DefaultUI, which was changed to simplify the menu given to the user when the software is started. In this case all options were removed apart from 'Start Experiment' and 'Exit'.

This last case requires additional explanation. As the Inference gametype was no longer used, the experiment instead runs using the default Deathmatch gametype. This presents few problems due to the construction of the level, as all weapons are removed and the spawn points are not usable by bots. However, by default Deathmatch includes a time limit of 20 minutes. As the experiment usually takes at least this much time to complete this is not acceptable.

To remove this limit an existing UnrealScript class, UTGameSettingsCommon, was modified. This class handles the game settings, and for some reason has the default game settings hard coded into it instead of presenting them in a Config file as with most other settings. This class was changed to force the default settings to have appropriate values (i.e. have no bots, no time limit and no goal score). This allows the main menu to serve simply to launch the experiment.

5.5 Critique of Implementation

With regards to requirements, the system fulfils all those necessary to conduct a basic Transitive Inference experiment. There were however several optional requirements that were not implemented.

Two of these involved taking timing information about the player's progress. These were both considered during implementation and ultimately rejected, both due to time constraints and some unsuitability issues. An array would likely be the best solution to store the large number of output variables that would be required in order to record the time taken for each trial. While Kismet does provide some limited array functionality it was found to integrate poorly with the Save Results action, and would also not transfer well to a Config file.

There were several other small issues during implementation as previously mentioned. These include the inability to change the number of colours, as well as the existence of a Head-up Display (HUD) that is mostly unused. While the option exists to remove the HUD entirely, this is not possible in this case as this would also remove the feedback messages given to the player. While it is certainly possible to remove elements of the HUD, a method for this was not found and the matter was dropped in favour of implementing more useful features.

Both the HUD and the ability to add additional colours would be an ideal focus of additional development time for the project. Overall though, the implementation of the system went according to plan and as needed by the requirements, and produces a satisfactory result, as demonstrated in the next chapter.

Chapter 6

Testing and Results

6.1 Introduction

Informal testing was performed during implementation of the system in order to help ensure that it performed as required. More formal testing was performed by running the experiment on several volunteers, both in order to show that the system may be used to gather experimental data and to find improvements and bugs that were missed during implementation.

As the tests performed during implementation were usually conducted quickly and informally while building the system they shall be mostly undocumented here, with two exceptions, discussed in the next two sections.

6.2 Testing Movement to the Triadic Phase

As part of personal testing the entire experiment was run several times in order to ensure that the flow of the experiment matched expectations. It was discovered while analysing the results produced by these tests that an incorrect sequence of trials was being given. In all cases an extra trial was given in the pair testing phase and one too few trials were given in the triadic testing phase. It was deemed most likely that this error was caused by some mistake in the Kismet sequence dealing with moving the player to the triadic testing phase.

After modifying the Kismet sequences several times, it was discovered that the Kismet did not determine that the player should be moved onto the triadic testing phase fast enough. The player was instead teleported again into the main test chamber to complete one final pair trial before at last being moved to the triadic test

chamber, despite a delay placed before the Teleport action. This was most likely caused by the check to decide if the pair testing phase was completed being called concurrently with other events, and relying on the Teleport delay to keep things going in the correct order. This was fixed by moving the check to take place directly before the Teleport action, ensuring that the check is completed before the player is teleported.

6.3 Testing Customisation of the Training Phase

As the training schedule is very customisable and allows the user to skip whatever stages of training desired, a more thorough testing scheme was developed. This scheme focused on the ability to skip stages of training, to help ensure that no matter the combination of stages used the experiment will run as required. This was done by removing each stage in order and also by removing adjacent stages, to make sure the system can account for this. The two extremes of 'all training stages' and 'no training stages' were also tested. Table 3 shows the order used for this, where the first row shows the stages of training (0-4) while the rest show whether that stage was included in the test:

Table 3 - Testing Scheme Used to Test User Customisation

0	**1**	**2**	**3**	**4**
Y	Y	Y	Y	Y
N	Y	Y	Y	Y
Y	N	Y	Y	Y
Y	Y	N	Y	Y
Y	Y	Y	N	Y
Y	Y	Y	Y	N
N	N	Y	Y	Y
Y	N	N	Y	Y
Y	Y	N	N	Y
Y	Y	Y	N	N

In all test cases the system performed as expected, smoothly moving to the next required stage and missing out any stages given 0 values in the Config file. In the final case, where all training stages are given 0, the experiment starts in the pair testing phase as expected. By extrapolation from these results it is believed that any combination of missing stages will behave appropriately.

6.4 Player Testing

The experiment was run using several volunteers, primarily to test usability. In total, five subjects attempted to complete the experiment. Each subject was presented with a laptop that had the game running, told the controls and given the following instructions:

"You will be presented with a series of doors, and you have to choose a door each time and go through it. You will be told on screen whether you have chosen the correct door. You must work out which door is the correct one in each case."

The first test subject went through the experiment using the training schedule described in Chapter 2 Table 1, as used by McGonigle and Chalmers in their 1984 experiment. The subject quickly grasped the ordering, and began complaining of boredom quickly, resulting in him trying to break the game by killing his character, first by attempting to trap his character in the doors as they closed, something that had been anticipated and made impossible during implementation.

He then uncovered a bug by using the 'Feign Death' command (A command built in to the Unreal Engine that causes the player's character to fall to the ground as if dead). This command was unknown during implementation and caused an unusual bug in that if the player uses this before teleporting it causes the character to actually die, prompting a respawn.

This bug along with a few others caused the experiment to be stopped so that the issues could be fixed. In the case of 'Feign Death' this was achieved by unbinding the appropriate key, causing the command to be inaccessible during play.

The issue of player boredom was also considered more thoroughly. Obviously some level of boredom is inevitable in an experiment such as this, but for the purposes of testing the system it was decided to reduce the length of the training phase for the remaining player tests. The revised training schedule is shown in Table 4, and essentially halves the number of pairs needed to pass the training, as well as eliminating Stage 3.

Of course, as the training schedule is customisable for each experiment and the data gathered is not to be used for any 'real' science the training schedule used for testing and the possible ramifications of this for the results will not be speculated on here. The revised schedule is merely mentioned to clarify the procedure used.

Table 4 - Revised Training Schedule Used During User Testing

Stage	Training and criteria
S0	Each pair in order (RB, BG, GY, YP) repeated until 4 consecutive trials correct for each pair. Reject if requires over 200 trials total.
S1	2 of each pair in order. Criterion: 16 consecutive trials correct. Reject if requires over 200 trials total.
S2	1 of each pair in order. Criterion: 8 consecutive trials correct. Reject if requires over 200 trials total.
S4	1 of each pair randomly ordered. Criterion: 16 consecutive trials correct. Reject if requires over 200 trials total.

The other four test subjects progressed through the game without encountering serious bugs, although several improvements were made to the system due to observation of the subjects playing the game and feedback from them afterwards.

One issue that came up for two of the subjects was a lack of experience playing First Person Shooter games on a computer, and a corresponding lack of experience with the control scheme. While they did learn the necessary controls within a few trials there were still some issues with 'missing the doors' that came up a few times as they played the game. While lack of familiarity with controls is not really possible to control, the doors were made wider in response to this so that they are easier to walk through. In addition the basic movement controls were made to appear on screen when the game starts so that the subject does not need to be told these verbally by those running the experiment.

Two other 'gameplay' issues came to light during these tests. Thanks to the model used for the doors, players would occasionally get 'caught' by the door when attempting to move through it and be forced to back away, causing confusion. This problem with fixed by changing the model of door to one without this possibility. Another problem that came up once was a player moving through a door and somehow not hitting the trigger behind it. In response to this, the triggers were made wider so that they could take up all of the space behind the doors.

6.5 Experimental Results

While detailed analysis of the results obtained during testing is not required, some discussion of the results obtained will be included here in order to demonstrate the kind of data that can be gathered using the system.

As previously stated, four test subjects attempted the experiment without encountering major bugs, all undergraduates from the University of Bath. The raw output of the UDKResults files for the subjects may be found in Appendix B. One subject did not complete training, failing during the final random stage of training.

Afterwards the subject reported having constructed a pattern where the correct colour for the previous pair affected the current pair, and attempted to make the feedback seen fit this pattern.

The other three subjects completed the experiment. Figure 8 shows the average correct results during pair testing for the three test subjects that completed the experiment:

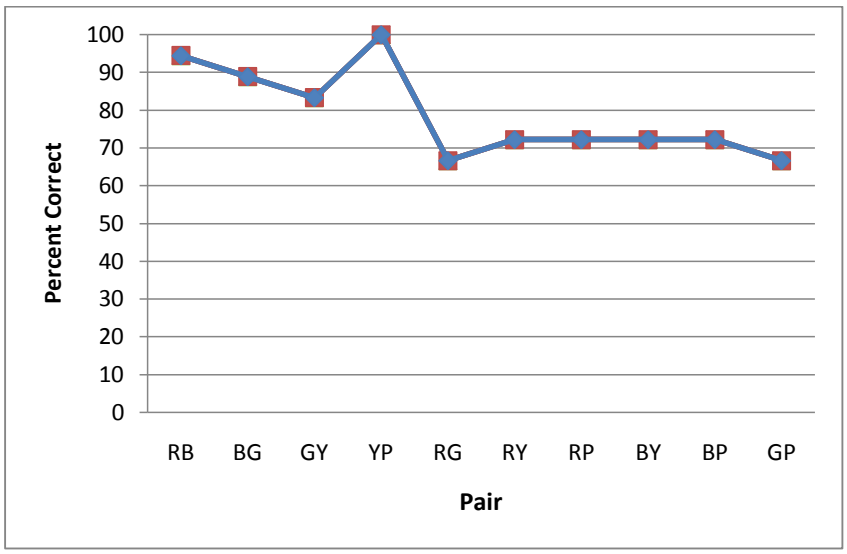

Figure 8. Average Percentage Correct Results for Each Pair (Training Pairs in Parenthesis)

It should be noted that two of the test subjects achieved 100% success rates on all pairs during the testing phase. The subject that didn't later reported that while they had constructed an ordering during the training phase they had difficulties in remembering this ordering during testing. The subject also thought that the ordering was cyclic, and that in a red-purple pair (a pair that never appears during training) that purple was the correct choice, while in reality purple is never correct.

Figure 9 shows the average correct results during triadic testing for the same three subjects. Again, two subjects achieved 100% success on all triads (although not the same two as during pair testing) while one subject made a few mistakes.

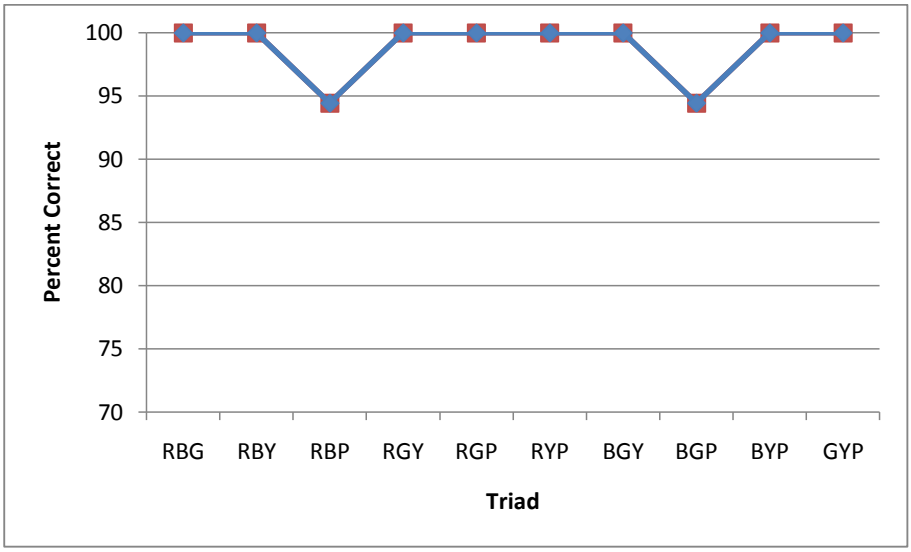

Figure 9. Average Percentage Correct Results for Each Triad

It is interesting to note the subjects on average performed better on triadic testing than on pair testing. The subject that performed badly during pair testing achieved a 100% success rate on the triads, and reported that they found it easier to remember the ordering of colours during this phase.

This could possibly be explained by only having five colours to choose from, only one of which does not contain an 'edge' object making it easier for the subject to choose correctly. This adds further to the idea of including the option for additional colours as an extension to the project.

Due to the small sample size more detailed analysis of the results would be largely speculation at this point, and is largely irrelevant from a computer science perspective. It is sufficient to see that the data can be collected and made use of.

Chapter 7

Conclusions

The aim of this project was to create a software system that allows the user to conduct repeatable, customisable experiments investigating transitive inference via a video game. This chapter summarises what has been achieved during the project, the significance of the work performed and discusses possible extensions and improvements that could be made to the system.

7.1 Project Summary

During the course of the Literature Review a thorough understanding of the inference problem was gained and several previous inference experiments were investigated and discussed. While only one of these experiments took place using a video game like interface (Siemann and Delius, 1993) they all provided useful perspectives on how experiments into inference were usually carried out. It was also decided at this time that the Unreal Development Kit was to be used to implement the project, after considering other options.

These previous experiments were then used as a model for a set of requirements. These requirements were generated based on the experimental procedures used, and where there were differences in procedure between experiments the requirements were chosen based on both discussions with Dr Bryson and personal decisions.

However, in many cases these differences in procedure could be accounted for by allowing the user to customise the parameters of the experiment as they wished. In these cases the requirements did not specify the relevant choices.

The preliminary design process was minimal and informal given previous lack of experience with the UDK and corresponding unfamiliarity with its capabilities. In

this case the design stage included several design decisions that proved unnecessary in the final system, such as the user of level streaming and the creation of a new gametype. On a more positive note, it was at this stage that a modular system of development for the Kismet sequences was chosen, and the subsequences needed were identified.

The experimental design itself was largely decided upon during the requirements gathering stage, and was closely modeled on that used by McGonigle and Chalmers in their 1984 experiment using young children. This accounted for much of the needed design, and the bulk of the project was a case of implementing it successfully.

Implementation was done using the UDK, as expected. After first creating the main test chamber, the inbuilt Kismet visual scripting language was used to create the necessary behavior for the experiment. This was done in a modular fashion as specified during the design stage by dividing the task into many smaller sequences, and then implementing and testing these sequences one at a time before moving onto the next. After the main test chamber was finished, the ability to incorporate triadic tests was implemented, as was the ability to customise several parameters used in the experiment via editing a Config file.

Testing was conducted on an informal basis throughout implementation to ensure that the system worked as required. More formal tests were conducted using volunteers, who were put through the experiment. These tests revealed several small bugs as well as a few usability issues that were soon fixed or improved. The experimental results from these test runs were briefly analysed and speculated upon.

7.2 Limitations and Possible Extensions

The system has a few limitations and issues that have come to light over the course of implementation and testing. These have in the most part been caused by either time constraints or unsuitability of the UDK. Most of these limitations could be dealt with or at least improved with further work as a possible extension of the project.

One feature that could be an important focus of further work is the option to include additional objects in the ordering, as in optional requirement 8.3. This feature was not implemented due to the fact that it would require much of the Kismet to be built again to allow for the additional combinations. This was deemed to be too time intensive, and a decision was made to focus on the core experiment using five colours.

Two other optional functional requirements were not met, requirements 7.2 and 7.3. These relate to gathering data about the time taken by the subject to complete each trial or to progress through each training stage respectively. While these were

considered during the implementation stage, both features were rejected due to limitations in the UDK, specifically concerning the use of arrays as explained in section 5.5, meaning that the time taken to implement them would be better spent elsewhere. Support for gathering timing data could potentially be added later, although this would likely be difficult.

Apart from the optional requirements there were a few minor issues that were not originally considered during the requirements stage, as well as some natural extensions of the project that came to light.

One such small issue that could be fixed in future is that of the superfluous HUD elements, such as the minimap and ammo counters. These proved rather difficult to remove without scrapping the HUD entirely, which was not really an option. While a solution no doubt exists, as custom HUDs are regularly included in games such as this, a satisfactory one could not be found and the issue was dropped, as the HUD does not impact the experiment in any meaningful way.

Another point is that the colours used in the experiment are hard coded via Kismet actions. Due to the way that colours are applied to the doors, easy user customisation of these via Config files or otherwise is not possible. Any desired change in colours (or other distinguishing features, such as shapes applied to the doors) used would require the user to access the Kismet and manually refer the Set Material action to a new source. This is not thought to be a major problem as the colours were arbitrarily chosen.

A possible extension is that while the editing of the UDKExperiment Config file provides the necessary customisation functionality, this is not ideal from a usability point of view. A better way of doing this would be to have these parameters available as settings in the main menu, allowing the user to customise them as needed from within the software. As this would require writing extensive UnrealScript classes, including a new front end menu, and would provide only a small usability benefit given that the user has to use a Config file to collect the results anyway this was not attempted during the project.

7.3 Critique of the Project

The project was overall a success, fulfilling all necessary requirements and most of the optional ones. The resulting software fulfills the aim of the project, and allows the user to conduct useful scientific experiments investigating Transitive Inference capabilities in humans. While the software does not provide all possible functionality, it runs the core five colour experiment well and with no discovered bugs, and allows the end user to customise the experiment as much as is feasible.

The UDK proved well suited to the task, especially the robust Kismet interface. This allowed the majority of the work to be completed without much written coding, an unexpected and not entirely welcome outcome, as more written code would have allowed an increased understanding of that side of game creation as well. The Kismet interface proved perfectly suitable however, and the creation of the custom Kismet Actions via UnrealScript was an interesting task, as outputting data to a human readable file is not readily supported by the UDK.

The initial design stage of the system was not very thorough, as at that point the full capabilities of the UDK were not well understood, and an approach of 'learn by doing' was thought to be more useful. Any future projects undertaken using the UDK would benefit from a more thorough and well planned design stage, allowing the implementation, and therefore the end result, to be more focused. In this project development time was lost by attempting to incorporate both level streaming and the new Inference gametype when neither was needed. A more thorough design stage would have prevented this, and the experience gained from this project would certainly help when planning and designing future projects.

Testing was mostly informal, as the final software does not have an extensive set of possible inputs to test, and was largely in aid of finding and fixing bugs. While it is generally foolish to claim that any software is bug free it is believed that the majority of bugs have been identified and fixed, as the software performs consistently well and produces accurate results.

The results obtained from the few experiments performed using volunteers were interesting. While two of the subjects performed near flawlessly, one failed during the random stage of training, perhaps indicating that the training schedule used for these experiments was inadequate for them. The near perfect performance of all three finishing test subjects during triadic testing perhaps shows that the option to include additional colours in the ordering would be a useful extension.

7.4 Closing Remark

This project was certainly a learning experience, and provided valuable insight and experience into elements of game design, as well as an appreciation of the psychological aspects and scientific experiments in general. While the opportunity for more written code would have been preferable from a learning standpoint, this was largely not needed for the software. The end result is a piece of software that could in future be used for real psychological research, with plenty of room for extension. Overall, the project was a success.

Bibliography

Bryson, J. and Leong, J. (2007) Primate errors in transitive 'inference': a two-tier learning model. Animal Cognition 10: 1-15

Davis, H. (1992) Transitive Inference in Rats (*Rattus norvegicus*). Journal of Comparative Psychology Vol. 106 No. 4 342-349

Greene, A.J., Spellman, B.A., Dusek, J.A., Eichenbaum, H.B. and Levy, W.B. (2001) Relational learning with and without awareness: Transitive inference using nonverbal stimuli in humans. Memory & Cognition 29: 893

McGonigle, B.O. and Chalmers, M. (1977) Are monkeys logical? Nature 267:694-696

McGonigle, B.O. and Chalmers, M. (1984) Are children any more logical than monkeys on the five term series problem? J Exp Child Psychol 37:355-377

McGonigle, B.O. and Chalmers, M. (1992) Monkeys are rational! Q J Exp Psychol 45B:189-228

Piaget, J. (1928) Judgment and reasoning in the child. Routledge and Kegan Paul, London

Siemann, M. and Delius, J.D. (1993) Implicit deductive reasoning in humans. Naturwissenshaften 80:364-366

Appendix A

User Documentation

This document briefly explains how to run a Transitive Inference experiment using the software created for this project. The experiment takes the form of a video game, where the subject has to select between pairs of doors of different colours. It is assumed that the reader has an understanding of Transitive Inference experiments.

A.1 Installation

To install the software, simply run the .exe file provided on the included CD and follow the instructions. At one point you will be asked to select an installation directory. Remember this directory, as it is where the results of the experiment will be placed when the experiment is concluded.

A.2 Starting an Experiment

To run the experiment with default settings, simply start the program, select 'Test Chamber' and then select 'Start Experiment'. The game will then start and present the subject with the main testing area.

The controls for the game are simple: User the arrow keys (or the WASD keys) to move the character around the level and through the doors.

A.3 The Default Experiment

The ordering used for the experiment is: Red>Blue>Green>Yellow>Purple. This ordering cannot be changed. The default training schedule used during the training phase is as follows:

Stage	Training and criteria
S0	Each pair in order (RB, BG, GY, YP) repeated until 10 consecutive trials correct. Reject if requires over 200 trials total.
S1	4 of each pair in order. Criterion: 32 consecutive trials correct. Reject if requires over 200 trials total.
S2	2 of each pair in order. Criterion: 16 consecutive trials correct. Reject if requires over 200 trials total.
S3	1 of each pair in order. Criterion: 30 consecutive trials correct. Reject if requires over 200 trials total.
S4	1 of each pair randomly ordered. Criterion: 25 consecutive trials correct. Reject if requires over 200 trials total.

During testing the subject will be given six blocks of the ten possible pairs, followed by six blocks of the ten possible triads.

This training schedule is based on McGonigle and Chalmers' 1984 experiment conducted on children, and depending on your requirements may be too extensive. That is why it is possible to change the training schedule.

A.4 Modifying the Experiment

To modify the training schedule, as well as the number of test blocks used, a certain text file must be modified. This file is named 'UDKExperiment' and is located at YourInstallationDirectory\UDKGame\Config. This file can be opened and modified using any text editor, such as Notepad. The file contains several parameters used in the experiment, all of which can be changed to suit your needs. These parameters are:

- PassZero - Indicating the number of successful consecutive trials required to move to the next pair in stage 0 of training.
- PassOne, PassTwo, PassThree and PassFour - All indicate the criterion needed to pass on to the next stage of training.
- FailZero, FailOne, FailTwo, FailThree and FailFour - These indicate the rejection criterion for each stage of training.

- NumPairOne, NumPairTwo and NumPairThree - These are slightly more complicated, and indicate the number of each pair used in their respective stages. For example, NumPairOne has a default value of 4, meaning that in stage 1 the subject is presented with 4 of each pair in order.
- TestBlocks and TriadBlocks - These indicate the number of blocks of tests given to the subject during the two testing phases.

Some of these values can be set to 0. If any of the Pass variables are set as 0, the corresponding stage of training will be skipped during the experiment. If any of the Fail variables are set as 0, no failure criteria will be used for that stage. If TraidBlocks is set as 0, no triadic tests will be performed. The NumPair variables and TestBlocks should always have a positive value.

To restore the default settings, simply delete the UDKExperiments file (NOT the DefaultExperiments file!).

A.5 Collecting the Results

At the conclusion of the experiment, the subject will be removed from the game and sent back to the main menu. At this point, a file is created holding the results from that subject. This file is called 'UDKResults' and is also located at YourInstallationDirectory\UDKGame\Config. This file should be moved to another location after each experiment, as it will be overwritten when the next experiment is run.

The values contained in this file give the results of the experiment, as follows:

- RBYes, BGYes etc - show the number of correct choices made for that pair during testing.
- RBNo, BGNo etc - show the number of incorrect choices made for that pair during testing.
- RBGYes, RBYYes etc - show the number of correct choices made for that triad during testing.
- RBGNo, RBYNo etc - show the number of incorrect choices made for that triad during testing.
- zerotrials, onetrials etc - show the number of trials needed to pass each stage of training.
- totaltrials - shows the total number of trials taken during the experiment, including training and testing.

Appendix B

Raw Results Output

Included in this Appendix are the four 'UDKResults' files obtained during user testing. These have been edited for space, in the actual files each result is on a new line.

B.1 Subject One

```
[Inference.SeqAct_SaveResults]
RBYes=5 BGYes=4 GYYes=3 YPYes=6 RGYes=0 RYYes=1 RPYes=1 BYYes=1
BPYes=1 GPYes=0
RBNo=1 BGNo=2 GYNo=3 YPNo=0 RGNo=6 RYNo=5 RPNo=5 BYNo=5 BPNo=5
GPNo=6
RBGYes=6 RBYYes=6 RBPYes=6 RGYYes=6 RGPYes=6 RYPYes=6 BGYYes=6
BGPYes=6 BYPYes=6 GYPYes=6
RBGNo=0 RBYNo=0 RBPNo=0 RGYNo=0 RGPNo=0 RYPNo=0 BGYNo=0 BGPNo=0
BYPNo=0 GYPNo=0
zerotrials=28 onetrials=33 twotrials=8 threetrials=0 fourtrials=22
totaltrials=211
```

B.2 Subject Two

```
[Inference.SeqAct_SaveResults]
RBYes=6 BGYes=6 GYYes=6 YPYes=6 RGYes=6 RYYes=6 RPYes=6 BYYes=6
BPYes=6 GPYes=6
```

```
RBNo=0 BGNo=0 GYNo=0 YPNo=0 RGNo=0 RYNo=0 RPNo=0 BYNo=0 BPNo=0
GPNo=0

RBGYes=6 RBYYes=6 RBPYes=6 RGYYes=6 RGPYes=6 RYPYes=6 BGYYes=6
BGPYes=6 BYPYes=6 GYPYes=6

RBGNo=0 RBYNo=0 RBPNo=0 RGYNo=0 RGPNo=0 RYPNo=0 BGYNo=0 BGPNo=0
BYPNo=0 GYPNo=0

zerotrials=27 onetrials=26 twotrials=13 threetrials=0 fourtrials=16
totaltrials=202
```

B.3 Subject Three

```
[Inference.SeqAct_SaveResults]

RBYes=0 BGYes=0 GYYes=0 YPYes=0 RGYes=0 RYYes=0 RPYes=0 BYYes=0
BPYes=0 GPYes=0

RBNo=0 BGNo=0 GYNo=0 YPNo=0 RGNo=0 RYNo=0 RPNo=0 BYNo=0 BPNo=0
GPNo=0

RBGYes=0 RBYYes=0 RBPYes=0 RGYYes=0 RGPYes=0 RYPYes=0 BGYYes=0
BGPYes=0 BYPYes=0 GYPYes=0

RBGNo=0 RBYNo=0 RBPNo=0 RGYNo=0 RGPNo=0 RYPNo=0 BGYNo=0 BGPNo=0
BYPNo=0 GYPNo=0

zerotrials=18 onetrials=17 twotrials=8 threetrials=0 fourtrials=0
totaltrials=244
```

B.4 Subject Four

```
[Inference.SeqAct_SaveResults]

RBYes=6 BGYes=6 GYYes=6 YPYes=6 RGYes=6 RYYes=6 RPYes=6 BYYes=6
BPYes=6 GPYes=6

RBNo=0 BGNo=0 GYNo=0 YPNo=0 RGNo=0 RYNo=0 RPNo=0 BYNo=0 BPNo=0
GPNo=0

RBGYes=6 RBYYes=6 RBPYes=5 RGYYes=6 RGPYes=6 RYPYes=6 BGYYes=6
BGPYes=5 BYPYes=6 GYPYes=6

RBGNo=0 RBYNo=0 RBPNo=1 RGYNo=0 RGPNo=0 RYPNo=0 BGYNo=0 BGPNo=1
BYPNo=0 GYPNo=0

zerotrials=27 onetrials=16 twotrials=8 threetrials=0 fourtrials=28
totaltrials=199
```

Appendix C

Code

Most of the implantation for this system was done via the Kismet visual scripting language, and will be summarised in Appendix D. This appendix contains full code listings for the two custom Kismet actions written for use in the game, as well as the three custom gametype classes that were written but not needed in the final product. One pre-existing UnrealScript class and several pre-existing Config files were modified as part of the project. These are included on the accompanying CD for completeness, as the modifications were rather minor in most cases.

C.1 SeqAct_SaveResults

```
class SeqAct_SaveResults extends SequenceAction

Config(Results); //declares that the class uses the Results Config file

var() config int RBYes; //variable declarations.

var() config int BGYes; var() config int GYYes; var() config int YPYes; var() config
int RGYes; var() config int RYYes; var() config int RPYes; var() config int BYYes;
var() config int BPYes; var() config int GPYes; var() config int RBNo; var() config
int BGNo; var() config int GYNo; var() config int YPNo; var() config int RGNo; var()
config int RYNo; var() config int RPNo; var() config int BYNo; var() config int BPNo;
var() config int GPNo; var() config int RBGYes; var() config int RBYYes; var() config
int RBPYes; var() config int RGYYes; var() config int RGPYes; var() config int
RYPYes; var() config int BGYYes; var() config int BGPYes; var() config int BYPYes;
var() config int GYPYes; var() config int RBGNo; var() config int RBYNo; var() config
int RBPNo; var() config int RGYNo; var() config int RGPNo; var() config int RYPNo;
var() config int BGYNo; var() config int BGPNo; var() config int BYPNo; var() config
int GYPNo; var() config int zerotrials; var() config int onetrials; var() config int
twotrials; var() config int threetrials; var() config int fourtrials; var() config
int totaltrials;
```

```
function Activated()

{

SaveConfig(); //when activated via Kismet, saves the input variables to the
//specified Config file.

}

defaultproperties {

ObjName="Save Results" //declares the name of the action in Kismet

ObjCategory="Output"

// the rest creates the input/output/variable links used in Kismet.

InputLinks(0)=(LinkDesc="In")

OutputLinks(0)=(LinkDesc="Out")

VariableLinks(1)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBYes", bWriteable=true,
PropertyName=RBYes) //creates the input links used in //Kismet.

VariableLinks(2)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGYes", bWriteable=true,
PropertyName=BGYes)

VariableLinks(3)=(ExpectedType=class'SeqVar_Int', LinkDesc="GYYes", bWriteable=true,
PropertyName=GYYes)

VariableLinks(4)=(ExpectedType=class'SeqVar_Int', LinkDesc="YPYes", bWriteable=true,
PropertyName=YPYes)

VariableLinks(5)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGYes", bWriteable=true,
PropertyName=RGYes)

VariableLinks(6)=(ExpectedType=class'SeqVar_Int', LinkDesc="RYYes", bWriteable=true,
PropertyName=RYYes)

VariableLinks(7)=(ExpectedType=class'SeqVar_Int', LinkDesc="RPYes", bWriteable=true,
PropertyName=RPYes)

VariableLinks(8)=(ExpectedType=class'SeqVar_Int', LinkDesc="BYYes", bWriteable=true,
PropertyName=BYYes)

VariableLinks(9)=(ExpectedType=class'SeqVar_Int', LinkDesc="BPYes", bWriteable=true,
PropertyName=BPYes)

VariableLinks(10)=(ExpectedType=class'SeqVar_Int', LinkDesc="GPYes", bWriteable=true,
PropertyName=GPYes)

VariableLinks(11)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBNo", bWriteable=true,
PropertyName=RBNo)

VariableLinks(12)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGNo", bWriteable=true,
PropertyName=BGNo)

VariableLinks(13)=(ExpectedType=class'SeqVar_Int', LinkDesc="GYNo", bWriteable=true,
PropertyName=GYNo)

VariableLinks(14)=(ExpectedType=class'SeqVar_Int', LinkDesc="YPNo", bWriteable=true,
PropertyName=YPNo)

VariableLinks(15)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGNo", bWriteable=true,
PropertyName=RGNo)

VariableLinks(16)=(ExpectedType=class'SeqVar_Int', LinkDesc="RYNo", bWriteable=true,
PropertyName=RYNo)

VariableLinks(17)=(ExpectedType=class'SeqVar_Int', LinkDesc="RPNo", bWriteable=true,
PropertyName=RPNo)

VariableLinks(18)=(ExpectedType=class'SeqVar_Int', LinkDesc="BYNo", bWriteable=true,
PropertyName=BYNo)
```

```
VariableLinks(19)=(ExpectedType=class'SeqVar_Int', LinkDesc="BPNo", bWriteable=true,
PropertyName=BPNo)

VariableLinks(20)=(ExpectedType=class'SeqVar_Int', LinkDesc="GPNo", bWriteable=true,
PropertyName=GPNo)

VariableLinks(21)=(ExpectedType=class'SeqVar_Int', LinkDesc="zerotrials",
bWriteable=true, PropertyName=zerotrials)

VariableLinks(22)=(ExpectedType=class'SeqVar_Int', LinkDesc="onetrials",
bWriteable=true, PropertyName=onetrials)

VariableLinks(23)=(ExpectedType=class'SeqVar_Int', LinkDesc="twotrials",
bWriteable=true, PropertyName=twotrials)

VariableLinks(24)=(ExpectedType=class'SeqVar_Int', LinkDesc="threetrials",
bWriteable=true, PropertyName=threetrials)

VariableLinks(25)=(ExpectedType=class'SeqVar_Int', LinkDesc="fourtrials",
bWriteable=true, PropertyName=fourtrials)

VariableLinks(26)=(ExpectedType=class'SeqVar_Int', LinkDesc="totaltrials",
bWriteable=true, PropertyName=totaltrials)

VariableLinks(27)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBGYes",
bWriteable=true, PropertyName=RBGYes)

VariableLinks(28)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBYYes",
bWriteable=true, PropertyName=RBYYes)

VariableLinks(29)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBPYes",
bWriteable=true, PropertyName=RBPYes)

VariableLinks(30)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGYYes",
bWriteable=true, PropertyName=RGYYes)

VariableLinks(31)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGPYes",
bWriteable=true, PropertyName=RGPYes)

VariableLinks(32)=(ExpectedType=class'SeqVar_Int', LinkDesc="RYPYes",
bWriteable=true, PropertyName=RYPYes)

VariableLinks(33)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGYYes",
bWriteable=true, PropertyName=BGYYes)

VariableLinks(34)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGPYes",
bWriteable=true, PropertyName=BGPYes)

VariableLinks(35)=(ExpectedType=class'SeqVar_Int', LinkDesc="BYPYes",
bWriteable=true, PropertyName=BYPYes)

VariableLinks(36)=(ExpectedType=class'SeqVar_Int', LinkDesc="GYPYes",
bWriteable=true, PropertyName=GYPYes)

VariableLinks(37)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBGNo", bWriteable=true,
PropertyName=RBGNo)

VariableLinks(38)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBYNo", bWriteable=true,
PropertyName=RBYNo)

VariableLinks(39)=(ExpectedType=class'SeqVar_Int', LinkDesc="RBPNo", bWriteable=true,
PropertyName=RBPNo)

VariableLinks(40)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGYNo", bWriteable=true,
PropertyName=RGYNo)

VariableLinks(41)=(ExpectedType=class'SeqVar_Int', LinkDesc="RGPNo", bWriteable=true,
PropertyName=RGPNo)

VariableLinks(42)=(ExpectedType=class'SeqVar_Int', LinkDesc="RYPNo", bWriteable=true,
PropertyName=RYPNo)
```

```
VariableLinks(43)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGYNo", bWriteable=true,
PropertyName=BGYNo)

VariableLinks(44)=(ExpectedType=class'SeqVar_Int', LinkDesc="BGPNo", bWriteable=true,
PropertyName=BGPNo)

VariableLinks(45)=(ExpectedType=class'SeqVar_Int', LinkDesc="BYPNo", bWriteable=true,
PropertyName=BYPNo)

VariableLinks(46)=(ExpectedType=class'SeqVar_Int', LinkDesc="GYPNo", bWriteable=true,
PropertyName=GYPNo)

}
```

C.2 SeqAct_Parameters

```
class SeqAct_Parameters extends SequenceAction

Config(Experiment); //declares that the class uses the Experiment Config file


var config int PassZero; //variable declarations.

var config int PassOne; var config int PassTwo; var config int PassThree; var config
int PassFour; var config int FailZero; var config int FailOne; var config int
FailTwo; var config int FailThree; var config int FailFour; var config int
NumPairOne; var config int NumPairTwo; var config int NumPairThree; var config int
TestBlocks; var config int TriadBlocks;

var() int PZ; var() int PO; var() int PTw; var() int PTh; var() int PF; var() int FZ;
var() int FO; var() int FTw; var() int FTh;b var() int FF; var() int NPO; var() int
NPTw; var() int NPTh; var() int TB; var() int TrB;


function Activated()

{
        PZ = PassZero;

        PO = PassOne;

        PTw = PassTwo;

        PTh = PassThree;

        PF = PassFour;

        FZ = FailZero;

        FO = FailOne;

        FTw = FailTwo;

        FTh = FailThree;

        FF = FailFour;

        NPO = NumPairOne;

        NPTw = NumPairTwo;

        NPTh = NumPairThree;

        TB = TestBlocks;

        TrB = TriadBlocks;

}
```

```
defaultproperties {

ObjName="Parameters" //declares the name of the action in Kismet

ObjCategory="Output"

// the rest creates the input/output/variable links used in Kismet.

InputLinks(0)=(LinkDesc="In")

OutputLinks(0)=(LinkDesc="Out")

VariableLinks(1)=(ExpectedType=class'SeqVar_Int', LinkDesc="PassZero",
bWriteable=true, PropertyName=PZ) //creates the input links used in Kismet.

VariableLinks(2)=(ExpectedType=class'SeqVar_Int', LinkDesc="PassOne",
bWriteable=true, PropertyName=PO)

VariableLinks(3)=(ExpectedType=class'SeqVar_Int', LinkDesc="PassTwo",
bWriteable=true, PropertyName=PTw)

VariableLinks(4)=(ExpectedType=class'SeqVar_Int', LinkDesc="PassThree",
bWriteable=true, PropertyName=PTh)

VariableLinks(5)=(ExpectedType=class'SeqVar_Int', LinkDesc="PassFour",
bWriteable=true, PropertyName=PF)

VariableLinks(6)=(ExpectedType=class'SeqVar_Int', LinkDesc="FailZero",
bWriteable=true, PropertyName=FZ)

VariableLinks(7)=(ExpectedType=class'SeqVar_Int', LinkDesc="FailOne",
bWriteable=true, PropertyName=FO)

VariableLinks(8)=(ExpectedType=class'SeqVar_Int', LinkDesc="FailTwo",
bWriteable=true, PropertyName=FTw)

VariableLinks(9)=(ExpectedType=class'SeqVar_Int', LinkDesc="FailThree",
bWriteable=true, PropertyName=FTh)

VariableLinks(10)=(ExpectedType=class'SeqVar_Int', LinkDesc="FailFour",
bWriteable=true, PropertyName=FF)

VariableLinks(11)=(ExpectedType=class'SeqVar_Int', LinkDesc="NumPairOne",
bWriteable=true, PropertyName=NPO)

VariableLinks(12)=(ExpectedType=class'SeqVar_Int', LinkDesc="NumPairTwo",
bWriteable=true, PropertyName=NPTw)

VariableLinks(13)=(ExpectedType=class'SeqVar_Int', LinkDesc="NumPairThree",
bWriteable=true, PropertyName=NPTh)

VariableLinks(14)=(ExpectedType=class'SeqVar_Int', LinkDesc="TestBlocks",
bWriteable=true, PropertyName=TB)

VariableLinks(15)=(ExpectedType=class'SeqVar_Int', LinkDesc="TriadBlocks",
bWriteable=true, PropertyName=TrB)

}
```

C.3 InferenceInfo

```
/*NOTE: This class is not used in the final system, though it is still referred to*/
class InferenceInfo extends UTGame
Config(UDKGame);

defaultproperties
{
        Acronym="I"
        MapPrefixes[0]="I"
        PlayerControllerClass=class'Inference.MyPlayerController'
        DefaultPawnClass=class'Inference.MyPawn'
        Name="Default__InferenceInfo"
}
```

C.4 MyPlayerController

```
/*NOTE: This class is not used in the final system, though it is still referred to*/
class MyPlayerController extends UTPlayerController;

event Possess(Pawn inPawn, bool bVehicleTransition)
{
        Super.Possess(inPawn, bVehicleTransition);
        SetBehindView(true);
}

defaultproperties
{
        Name="Default__MyPlayerController"
}
```

C.5 MyPawn

```
/*NOTE: This class is not used in the final system, though it is still referred to*/
class MyPawn extends UTPawn;

defaultproperties
{
        Begin Object Name=WPawnSkeletalMeshComponent
                bOwnerNoSee=false
        End Object
        Name="Default__MyPawn"
}
```

Appendix D

Viewing Kismet and Modifying the Software

The majority of the work done on this project was using the Kismet visual scripting language included with the UDK. As this is difficult to represent in text format, and screenshots provide only a limited and confusing view of the Kismet, this appendix exists as a brief guide on installing and viewing the code files provided using the UDK. This will allow the reader to both view the complete Kismet sequence created for the software and also give them the ability to modify the software in any way they wish.

Install the Unreal Developers Kit (available for free from www.udk.com) and open the I-TestChamber.udk file provided on the accompanying CD. The Kismet sequence can be accessed by clicking on the green 'K' icon in the top toolbar.

In order to get full functionality in the UDK, the accompanying code files must be copied into the reader's UDK installation directory as follows:

The Config files must be copied into the \UDKGame\Config folder, overwriting those found there as necessary.

The UTGameSettingsCommon.uc file must be placed in the:
\Development\Src\UTGame\Classes folder, overwriting the existing class.

Two new folders must be created as \Inference\Levels and placed in the existing \UDKGame\Content\ folder. The provided I-TestChamber file must be placed in Levels. i.e. \UDKGame\Content\Inference\Levels\I_TestChamber

Two more new folders must be created as \Inference\Classes and placed in the existing \Development\Src folder. The five additional .uc files must be placed in Classes. i.e. \Development\Src\Inference\Classes\SeqAct_SaveResults.uc

This should ensure that the system works as needed. Remember, this process is only required if the reader wishes to modify or extend the software in some way. The specific version of the UDK used for this project was the February 2011 build, and some issues may be created if a different build is used.